BASIC CONCEPTS OF THE JAVASCRIPT

Boris Bocharov, Maria Voevodina,
Nataliia Braterska, Anastasiia Dashkovska

JavaScript is programming language for Web. The vast majority of websites use JavaScript, and all modern web browsers - for desktops, game consoles, electronic tablets and smartphones - include a JavaScript interpreter, which makes JavaScript the most widely used programming language that have ever existed in history. JavaScript is one of three technologies that any web developer should know [1,2]:

- HTML markup language that allows you to define the content of web pages,
- CSS style language, which allows you to define the appearance of web pages,
- The programming language JavaScript, which allows you to determine the behavior of web pages.

If you know other programming languages, you may find information useful that JavaScript is a high-level, dynamic, untyped and interpreted programming language that is well suited for programming in object-oriented and functional styles.

Its JavaScript syntax inherited from the Java language, its first-class functions - from the Scheme language, and the mechanism of inheritance on the basis of prototypes - from the language of Self.

The name of the language "JavaScript" can be misleading. With the exception of surface syntactic similarity, JavaScript is completely different from the Java programming language.

JavaScript has long outgrown the framework of the scripting language, turning into a reliable and effective universal programming language.

JavaScript was created in the company Netscape at the dawn of the Web. The name "JavaScript" is a trademark registered by Sun Microsystems (now Oracle), and is used to describe the implementation of the language created by Netscape (now Mozilla). Netscape has introduced a language for standardization of the European Computer Manufacturer's Association (ECMA), but because of

legal trademark problems, the standardized version of the language has received a somewhat awkward name, ECMAScript.

Due to the same legal problems, the language version from Microsoft received the official name "JScript". However, in practice, all these implementations are usually called JavaScript

During the last decade, all web browsers provided implementation of version 3 of the ECMAScript standard, and in fact, developers did not need to think about version numbers: the language standard was stable, and its implementation in web browsers was largely compatible.

Recently, a new important version of the standard language called ECMAScript 5 has released.

The fourth version of the ECMAScript standard was developed for many years, but because of too ambitious goals it was never released.

Google is developing its JavaScript interpreter called V8. It is based on the version of node js.

A Brief Overview of the JavaScript Language

```
// Everything that follows two slashes is a comment.
// Attentively read the comments: they describe the JavaScript
code.
// Variables are declared using the keyword var:
var x; // Declaring a variable named x
// Assign values to variables using the =
x = 0; // Now the variable x has the value 0
// x => 0: In expressions, the name of the variable is replaced by
its value.
// JavaScript supports values of different types
x = 1; // Numbers.
x = 0.01; // Integer and real numbers are represented by one type.
x = "hello world"; // Strings of text in quotes.
x = 'JavaScript'; // Strings can also be enclosed in apostrophes.
x = true; // Boolean values.
```

```
x = false; // Another Boolean value.
x = null;  // null is a special value, indicating as "no value".
x = undefined; // The value of undefined is similar to null
```

Two other very important types of data that can be manipulated by JavaScript programs are objects and arrays. They will be considered later.

```
// The most important type of data in JavaScript are objects.
// An object is a collection of name/value pairs or a mapping of a
string to a value.
var book = {    // The objects are enclosed in braces.
topic: "JavaScript", // The property "topic" is set to
"JavaScript".
fat: true            // The property "fat" is set to true.
};                   // The braces mark the end of the object.
// Access to object properties using ‘.’ or ‘[]’ :
book.topic   // => "JavaScript"
book["fat"]  // => true: another way to get the property value.
book.author = “Ivanov”; // Create a new property by assignment.
book.contents = {};       // {} - empty object
// JavaScript supports arrays (lists with numeric indices) of
values:
var primes = [2, 3, 5, 7];// An array of 4 values is limited to [
and ].
primes[0]    // => 2: the first element (with the index 0) of array
primes.length       // => 4: number of elements in the array.
primes[primes.length-1]   // => 7: the last element of the array.
primes[4] = 9;            // Add a new element assignment.
primes[4] = 11;     // Or change the value of an existing item.
var empty = [];     // [] – empty array without ellements.
empty.length        // => 0
// Arrays and objects can store other arrays and objects:
```

```
var points = [      // Array with 2 elements.
{x:0, y:0},  // Each element is an object.
x:1, y:1}
];
var data = {         // Object with 2 properties.
 trial1: [[1,2], [3,4]], // The value of each property is an array.
 trial2: [[2,3], [4,5]] // The elements of the array are arrays.
};
```

Syntactic constructs that contain lists of array elements in square brackets or display object properties in values within braces are often called initialization expressions.

An expression is a phrase in the JavaScript language that can be calculated to get a value. For example, use '.' and '[]' to reference the value of the property of an object or array element is an expression.

The most common way of generating expressions in JavaScript is to use operators.

```
// Operators perform actions with values (operands) and reproduce
the new value.
// The most commonly used are arithmetic operators:
3 + 2        // => 5: addition
3 - 2        // => 1: subtraction
3 * 2        // => 6: multiplication
3 / 2        // => 1.5: division
points[1].x - points[0].x // => 1: you can use more complex
operands
"3" + "2"           // => "32": + adds numbers, combines strings
// In JavaScript, there are some abbreviated forms of arithmetic
operators
var count = 0;      // Declaration of variable
count++;            // Increase the value of the variable by 1
count--;            // Decreasing the value of the variable by 1
```

```
count += 2;         // Add 2: the same as count = count + 2;
count *= 3;         // Multiply by 3: same as count = count * 3;
count               // => 6: names of variables themselves are
expressions
// Comparison operators allow you to check two values for equality
// or inequality, find out which value is less or more, etc.
// They return true or false.
var x = 2, y = 3;   // Signs '=' perform assignment, not comparison
x == y              // => false: equality
x!= y               // => true: inequality
x < y               // => true: less
x <= y              // => true: less or equally
x > y               // => false: more
x >= y              // => false: more or equally
"two" == "three"    // => false: two different strings
"two" > "three"     // => true: when sorting alphabetically the
strings
// "tw" more than "th"
false == (x> y) // => true: false equally false
// Logical operators combine or invert logical values
(x == 2) && (y == 3)      // => true: both comparisons are true.
&& - "and"
(x > 3) || (y < 3)        // => false: both comparisons are false.
|| - "or"
! (x == y) // => true: ! inverts a Boolean value
```

A function is a named and parameterized block of JavaScript code that is defined once, and can be used multiple times.

Here are some examples of simple functions:

```
function plus1(x) { // Define a function named "plus1" and with the
parameter "x"
```

```
      return x+1;  // Return value 1 more than received
} // Functions are enclosed in braces
plus1(y)     // => 4: y has a value of 3, so this call will return
3 + 1
var square = function(x) { // Functions can be assigned to
variables
return x*x;  // Calculate the value of the function
};     // A semicolon marks the end of the assignment.
square(plus1(y))    // => 16: call two functions in one expression
```

When you combine functions with objects, you get the following methods:

```
// The functions assigned to the properties of objects are called
methods.
// All objects in JavaScript have methods:
var a = [];   // Create an empty array
a.push(1,2,3);// The push () method adds elements to the array
a.reverse();  // Another method: rearranges the elements in the
reverse order
// You can define your own methods.
// The keyword "this" refers to the object in which the method is
defined: in this case, the array points.
points.dist = function() { // Method for calculating the distance
between points
var p1 = this[0];   // The first element of the array, relative to
which the method is called
var p2 = this[1];         // The second element of the "this"
var a = p2.x-p1.x;        // Difference of coordinates X
var b = p2.y-p1.y;        // Difference of coordinates Y
return Math.sqrt(a*a + b*b); // Math.sqrt() calculates the square
root
};
```

```
points.dist() // => 1.414: distance between 2 points
```

Consider several functions that demonstrate the application of the most commonly used JavaScript control instructions:

```
// JavaScript has conditional instructions and instructions of
cycles syntactically similar to similar instructions C, C ++, Java
and other languages.
function abs(x){ // The function that calculates the absolute value
if (x >= 0){ // instructions if …
return x;    // executes this code if the comparison yields true.
}     // the end of instructions if
else{        // An optional else clause executes its code,
return -x;          // if the comparison evaluates to false.
} // Braces can be omitted if the sentence contains 1 statement.
function factorial(n) {   // The function that calculates the
factorial
    var product = 1;      // Start with a product equal to 1
    while(n > 1) {  // Repeat the instructions in {} while
expressing in () true
        product *= n;     //The shortened form of the expression
product = product*n
        n--;              // The abbreviated form of the
expression n = n – 1
    }               // the end of cycle
    return product;
}
factorial(4) // => 24: 1*4*3*2
function factorial2(n) { // Another version that uses a different
cycle
var i, product = 1;       // Start from 1
for(i=2; i <= n; i++)     // i automatically increases from 2 to n
```

```
product *= i;              // Perform in each cycle {} can be
omitted,
return product; // return factorial
}
factorial2(5) // => 120: 1*2*3*4*5
```

JavaScript is an object-oriented language, but the object model used in it differs radically from the model used in most other languages.

Below is a very simple example demonstrating the definition of the JavaScript class for representing points on a plane. Objects that are instances of this class have a single method r () that computes the distance between a given point and the origin:

```
// Define a constructor function to initialize a new object Point
function Point(x,y) { //By agreement, the name of the constructors
                      // begins with the capital letter
this.x = x;  // this - reference to the initialized object
this.y = y;  // Save arguments in object properties
} // Nothing is returned
// To create a new instance, you must call the constructor function
with the keyword "new"
var p = new Point (1, 1); // Point on the plane with coordinates
(1,1)
// Methods of objects Point are determined by assigning functions
// to the properties of the prototype object associated with the
constructor function.
Point.prototype.r = function () {
// Return the square root of x2 + y2
// this is a Point object, against which the method is invoked.
    return Math.sqrt( this.x * this.x +  this.y * this.y );
};
// Now object p of type Point (and all subsequent Point objects)
//inherits the method r()
```

```
p.r () // => 1.414 ...
```

Lexical Structure

The lexical structure of a programming language is a set of elementary rules that determine how programs are written in that language. This is a low-level syntax for the language; it defines the kind of variable names, the symbols used to denote comments, and how one instruction separates from the other.

Character set

When writing programs in JavaScript, the Unicode character set is used. Unicode is a superset of ASCII and Latin-1 encodings and supports almost all written languages on the planet. The ECMAScript 3 standard requires that JavaScript implementations support the Unicode standard version 2.1 or higher, and the ECMAScript 5 standard requires that implementations provide support for the Unicode version 3 or higher.

Case sensitivity

JavaScript is a language that is case sensitive. This means that keywords, variable and function names, and any other language identifiers must always contain the same set of uppercase and lowercase letters. For example, the keyword while must be typed as "while", not "While" or "WHILE". Similarly online, Online, OnLine and ONLINE are names of four different variables.

Note, however, that HTML markup language (unlike XHTML) is not case sensitive. As HTML and client JavaScript are tightly coupled, this difference can lead to confusion. Many JavaScript objects and their properties have the same names as HTML tags and attributes, which they denote. However, if in HTML these tags and attributes can be typed in any register, then in JavaScript they usually should be typed in lowercase letters. For example, the attribute onclick of the event handler is most often set in HTML as onClick, however, in JavaScript code (or in an XHTML document), it must be marked as onclick.

Spaces, line breaks and format control characters

JavaScript ignores spaces that can be present between tokens in the program. In addition, JavaScript also for the most part ignores line feed characters. Therefore, spaces and newlines can be used without restriction in the source code of the programs for formatting and giving them a human-readable appearance. In addition to the usual space character (\ u0020), JavaScript also recognizes as whitespace the following characters: tab (\ u0009), vertical tab (\ u000B), format translation (\ u000C), non-breaking space (\ u00A0), byte order marker (\ uFEFF), as well as all Unicode characters belonging to the category Zs. The following characters are recognized by the Javascript interpreter as end-of-line characters: line feed (\ u000A), carriage return (\ u000D), line separator (\ u2028) and paragraph separator (\ u2029). A sequence of carriage return and linefeed characters is interpreted as a single line termination character. Unicode characters that control the format (category Cf), such as RIGHT-TO-LEFT MARK (\ u200F) and LEFT-TO-RIGHT MARK (\ u200E), control the visual representation of the text in which they are present. They are important for the correct display of text in some languages and are valid in JavaScript comments, string literals, and regular expression literals, but not in identifiers (such as variable names) defined in JavaScript programs. The exceptions are ZEROWIDTH JOINER (\ u200D) and ZERO WIDTH NON-JOINER (\ u200C), which you can use in identifiers, provided that they are not the first characters of identifiers. As noted above, the byte order control character (\ uFEFF) is interpreted as a whitespace character.

Shielded Unicode sequences can also appear in comments, but since comments are ignored, in this context they are treated as a sequence of ASCII characters and are not interpreted as Unicode characters.

Normalization

Unicode allows you to encode the same character in several ways. These presentation methods provide the same display in a text editor, but they have different binary codes and are considered different from the computer's point of view. The Unicode standard specifies the preferred encoding methods for all characters and specifies a normalization procedure for converting the text to a canonical form suitable for comparison. JavaScript interpreters believe that the interpreted code has already been normalized, and does not attempt to normalize identifiers, strings, or regular expressions.

Comments

JavaScript supports two ways of commenting. Any text between // and the end of a line is treated as a comment and ignored by JavaScript. Any text between the / * and * / characters is also treated as a comment. These comments can consist of several lines, but can not be nested. The following lines are valid JavaScript comments:

```
// This is a one-line comment.
/ * This is also a comment * / // and this is another comment.
/ *
* This is another comment.
* It is located in several lines.
* /
```

Literals

A literal is the value specified directly in the text of the program. The following are examples of different literals:

```
12 // Number twelve
1.2 // The number one is two-tenths
"hello world" // Text line
'Hi' // Another line
true // Boolean
false // Other Boolean value
/ javascript / gi // The literal of the "regular expression" (for
searching by pattern)
null // Empty object
Below are more complex expressions that can serve as literals for
arrays and objects:
{x: 1, y: 2} // Object initializer
[1,2,3,4,5] // Array initializer
```

Identifiers and reserved words

An identifier is just a name. In JavaScript, identifiers act as the names of variables and functions, as well as labels for some loops. Identifiers in JavaScript must begin with a letter, with an underscore (_) or a dollar sign ($).Any letters, numbers, underscores or dollar signs can follow. (A digit can not be the first character, since then it will be difficult for an interpreter to distinguish identifiers from numbers). Examples of valid identifiers:

```
i
my_variable_name
v13
_dummy
$str
```

For compatibility and ease of editing, only ASCII characters and numbers are usually used to compose identifiers. However, JavaScript allows the use of letters and digits from the full set of Unicode characters in identifiers. (Technically, the ECMAScript standard also allows Unicode characters from the Mn, Mc and Pc categories in the identifiers, provided they are not the first characters of the identifiers.) This allows programmers to give names to variables in their native languages and use mathematical symbols in them.

JavaScript reserves a number of identifiers that play the role of keywords of the language itself. These keywords can not serve as identifiers in programs:

1. break
2. case
3. catch
4. continue
5. debugger
6. default
7. delete
8. do
9. else
10. false
11. finally
12. for
13. function

14. if
15. in
16. instancejof
17. new
18. null
19. return
20. switch
21. this
22. throw
23. true
24. try
25. typeof
26. var
27. void
28. while

Data Types, Values, and Variables

During operation, computer programs manipulate values, for example the number 3.14 or the text "Hello World". Data types The types of values that can be represented and processed in a programming language are known as data types, and one of the most fundamental characteristics of any programming language is the set of data types it supports.

When a program needs to save a value to use it later, this value is assigned (or stored in) a variable. A variable defines a symbolic name for a value and provides the ability to get this value by name. The principle of the action of variables is another fundamental characteristic of any language.

The types of data in JavaScript can be divided into two categories: simple types and objects. The category of simple types in the JavaScript language includes numbers, text strings (which are usually called just strings) and logical (or Boolean) values.

The special values “null” and “undefined” are elementary values, but they do not apply to numbers, or to strings, or to boolean values. Each of them defines only one value of its own special type.

Any value in JavaScript that is not a number, a string, a Boolean value, or a special value, “null” or “undefined”, is an object.

An object (that is, a member of an object data type) is a collection of properties, each of which has a name and value (either a simple type, such as a number or string, or an object type).

Numbers

Unlike many programming languages, JavaScript does not distinguish between integers and real values. All numbers in JavaScript are represented by real values (floating point).

To represent numbers in JavaScript, a 64-bit format is used, defined by the IEEE 754.1 standard.

This format is known to programmers in the Java language as a double type. The same format is used to represent numbers of type double in most modern implementations of C and C ++.

This format is able to represent numbers in the range of ±1.7976931348623157x10308 to ± 5x10-324.

The format of the representation of real numbers in JavaScript allows you to accurately represent all integers from -9007199254740992 (-253) to 9007199254740992 (253) inclusive. For integer values outside this range, precision in the lower order bits may be lost.

It should be noted that some operations in JavaScript (such as accessing array elements by indices and bit operations) are performed with 32-bit integer values.

The number that is directly in the program in the JavaScript language is called a numeric literal.

Arithmetic operations in JavaScript

The processing of numbers in the JavaScript language is performed using arithmetic operators. The number of such operators includes: the addition operator +, the subtraction operator -, the multiplication operator *, the division operator / and the modulo operator% (returns the remainder of the division). A full description of these and other operators can be found in the reference.

In addition to these simple arithmetic operators, JavaScript supports more complex mathematical operations, using functions and constants available as properties of the Math object.

Arithmetic operations in JavaScript do not raise an error in case of overflow, loss of significant digits or division by zero. If the result of the arithmetic operation is greater than the largest representable value (overflow), the special value "infinity", which in JavaScript is denoted as Infinity, is returned. Similarly, if the absolute value of the negative result is greater than the largest representable value, the value "negative infinity" is returned, which is designated as -Infinity. These special values denoting infinity behave exactly as one would expect: adding, subtracting, multiplying, or dividing infinity by any value results in an infinite (possibly with opposite sign).

Loss of significant digits occurs when the result of the arithmetic operation is closer to zero than the minimum possible value. In this case, the number returns 0. If the loss of significant digits occurs in a negative result, a special value, known as "negative zero," is returned.

Division by zero is not considered an error in JavaScript: in this case, simply returns infinity or negative infinity. However, there is one exception: the operation of dividing zero by zero does not have a well-defined value, so the special value "not-a-number", which is denoted as NaN, is returned as the result of such an operation. The NaN value is also returned when trying to divide infinity into infinity, extract the square root of a negative number, or perform an arithmetic operation with non-numeric operands that can’t be converted to numbers.

Text

A string is an unchanged, ordered sequence of 16-bit values, each of which usually represents a sign of Unicode.

The strings in JavaScript are the data type used to represent the text. The length of a string is the number of 16-bit values contained in it. The numbering of characters in strings (and elements in arrays) in the JavaScript language starts from zero: the first 16-bit value is in position 0, the second in position 1, etc.

An empty string is a string whose length is 0. In JavaScript, there is no special type for representing a single element of a string. To represent a single 16-bit value, simply use a string with a length of 1.

To include a literal string in a JavaScript program, it's enough just to enclose the string's characters in paired single or double quotes ('or'). Double-quoted characters can be contained in strings delimited by single-quote characters, and single-quote characters in strings delimited by double-quote characters.

The backslash (\) has a special purpose in JavaScript strings. Along with the characters following it, it denotes a character that is not representable within the string in other ways. For example, \ n is an escape sequence that denotes a line feed character.

Another example is the sequence \', which stands for the single-quote character.

Work with strings

One of the built-in JavaScript features is the ability to concatenate strings. If the + operator is applied to numbers, they are added up, and if to the rows - they are combined, with the second line added to the end of the first.

For example:

```
msg = "Hello," + "world"; // Get the string "Hello, world"
greeting = "Welcome to my homepage," + "" + name;
```

To determine the length of a string-the number of 16-bit values contained in it-the length property is used. For example, the length of the string s can be obtained as follows:

```
s.length
```

Besides, in addition to the length property, strings have many methods (see the reference section for more information):

```
var s = "hello, world" // Let's start with the same text.
s.charAt (0) // => "h": the first character.
s.charAt (s.length-1) // => "d": last character.
s.substring (1,4) // => "ell": the 2nd, 3rd and 4th characters.
s.slice (1,4) // => "ell": the same
s.slice (-3) // => "rld": last 3 characters
s.indexOf ("l") // => 2: position of the first character l.
```

```
s.lastIndexOf ("l") // => 10: position of the last character l.
s.indexOf ("l", 3) // => 3: position of the first character "l",
the next
                              // for 3 characters in a line
s.split (",") // => ["hello", "world"] breaks into substrings
s.replace ("h", "H") // => "Hello, world": replaces all occurrences
of substring
s.toUpperCase () // => "HELLO, WORLD"
```

Date and time

In the base JavaScript language, there is a Date () constructor to create objects that represent the date and time. These Date objects have methods for performing simple calculations involving dates. The Date object is not a fundamental data type, like numbers. A full description can be found in the reference section.

```
var then = new Date (2010, 0, 1); // First day of the first month
of 2010
var later = new Date (2010, 0, 1, 17, 10, 30);      // The same
date, at 17:10:30
                                                    // local time
var now = new Date (); // Current date and time
var elapsed = now - then; // Difference of dates: interval in
milliseconds
later.getFullYear () // => 2010
later.getMonth () // => 0: months count starts from zero
later.getDate () // => 1: The days count starts with a unit
later.getDay () // day of the week. 0 - resurrection, 5 - spots.
later.getHours () // 17 hours local time
later.getUTCHours () // hours by UTC; depends on the time zone
later.toString () // "Fri Jan 01 2010 17:10:30 GMT + 0300"
later.toUTCString () // "Fri, 01 Jan. 2010 14:10:30 GMT"
```

```
later.toLocaleDateString () // "January 1, 2010"
later.toLocaleTimeString () // "17:10:30"
```

Null and undefined

The null keyword in JavaScript language has a special purpose and is usually used to indicate the absence of a value. The typeof operator for null returns the string "object", which means that null is a special "empty" object. However, in practice, the value of null is usually considered to be the only member of a proprietary type and can be used as a sign of no value, such as a number, string, or object.

In the JavaScript language, there is one more value indicating that there is no value. The value is undefined, indicating the complete absence of any value. It returns when you access a variable that has never been assigned a value, as well as a nonexistent property of the object or array element. In addition, the value of undefined is returned by functions that do not have a return value, and is assigned to function parameters for arguments that were not passed on the call.

Boolean values

A logical value indicates the truth or falsity of something. A logical data type has only two valid logical values. These two values are represented by literals true and false.

Boolean values are usually the result of comparison operations performed in JavaScript programs.

For example:

```
a = = 4
```

This expression checks whether the value of a is equal to the value of 4. If so, the result of this comparison is a logical value of true. If the value of a is not 4, the result of the comparison is false.

Boolean values are commonly used in JavaScript control constructs. For example, the if / else statement in JavaScript performs one action if the Boolean value is true, and another action if false. Typically, a comparison that creates a Boolean value is directly combined with the instruction in which it is used. The result looks like this:

```
if (a = = 4)
      b = b + 1;
else
     a = a + 1;
```

Equality and inequality operators

The = = and = = = operators check two values for a match, using two different match definitions. Both operators accept operands of any type and return true if their operands match, and false if they are different. The = = = operator, known as an identity operator, checks two operands for "identity", guided by a strict definition of a match. The = = operator, the equality operator, checks whether its two operands are equal in accordance with a less strict definition of match that allows type conversions.

In the JavaScript language, the operators =, = = and = = = are supported. Make sure that you understand the difference between the assignment, equality and identity operators.

Be careful and apply the right operators when developing your programs! It is very tempting to call all three operators "equal", but in order to avoid confusion it is better to read the statement = how "it turns out" or "assigned", the operator = = read as "equal", and the word "identically" means the operator = = =.

Operators != And ! = = mean "not equal" and "not identical".

The identity operator = = = computes the values of its operands, and then compares the two values, without converting the types, according to the following rules:

1. If two values have different types, they are not identical.
2. If both operands are null or undefined, they are identical.
3. If both operands are a Boolean value true or both are Boolean false, they are identical.
4. If one or both of the values are NaN, they are not identical. The value of NaN is never identical to any value, even to yourself! To check whether the value of x is NaN, use the expression x! = = x. The value of NaN is the only one for which such a test will return true.

5. If both values are numbers with the same value, they are identical. If one operand is 0 and the other is -0, they are also identical.
6. If both values are strings and contain the same 16-bit values in the same positions, they are identical. If the lines differ in length or content, they are not identical. Two lines can have the same meaning and look the same on the screen, but contain different sequences of 16-bit values. The JavaScript interpreter does not normalize Unicode characters, so these pairs of strings are not considered equal = = = and = = neither equal nor identical.
7. If both values refer to the same object, array, or function, then they are identical. If they refer to different objects (arrays or functions), they are not identical, even if both objects have identical properties.

The equality operator = = is similar to the identity operator, but it uses less stringent rules. If the values of the operands are of different types, it performs type conversion and tries to perform a comparison:

1. If two values are of the same type, they are checked for identity, as described above. If the values are identical, they are equal; if they are not identical, they are not equal.
2. If two values do not refer to the same type, the = = operator can still consider them equal. The following rules and type conversions are used:
3. If one value is null and the other is undefined, then they are equal.
4. If one value is a number and the other is a string, the string is converted to a number and a comparison is made with the converted value.
5. If any value is true, it is converted to 1 and the comparison is executed again. If any value is false, it is converted to 0 and the comparison is executed again.
6. If one of the values is an object and the other is a number or string, the object is converted to a simple type and the comparison is executed again. The object is converted to a value of a simple type either using its toString () method, or using its valueOf () method. The built-in JavaScript base class classes first try to perform the valueOf () conversion, and then toString (), except for the Date class, which always performs the toString () conversion. Objects that are not part of the basic JavaScript can transform themselves into values of simple types in the way defined by their implementation.

7. Any other combination of values is not equal.

As an example of equality testing, consider the comparison:

```
"1" = = true
```

The result of this expression is true, i.e. these different looking values are actually equal. The boolean value true is converted to the number 1, and the comparison is executed again. Then, the string "1" is converted to the number 1. Since both numbers now match, the comparison operator returns true.

Practical Work. The simplest calculations.

Write a program that tests the formulas of abridged multiplication.

For all formulas, it is necessary to program the calculation of the right and left parts of the formula.

Add the necessary code to the next program.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title>John Smith</title>
<script>
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
```

```
}
</script>
</head>
<body>
<div id="my">conteiner</font></div>
<script>
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
cnt.out("<font color='#0000FF' size='+3'>The simplest calculations
</font><br>");
a = 2;
b = 7;
cnt.add("a = "+a+"<br>");
cnt.add("b = "+b+"<br>");
c1 = (a+b)*(a+b);
c2 = a*a + 2*a*b + b*b;
cnt.add("(a + b)^2= " + c1+"<br>");
cnt.add("a^2 + 2ab + b^2 = " + c2+"<br>");
cnt.add("<br><h1><i> Add code for other sections </i></h1><br>");
</script>
</body>
</html>
```

Formulas for squares

$(a + b)^2 = a^2 + 2ab + b^2$
$(a – b)^2 = a^2 – 2ab + b^2$
$a^2 – b^2 = (a – b)(a + b)$
$(a + b + c)^2 = a^2 + b^2 + c^2 + 2ab + 2ac + 2bc$

Formulas for cubes

$(a + b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$
$(a – b)^3 = a^3 – 3a^2b + 3ab^2 – b^3$

$a^3 + b^3 = (a + b)(a^2 – ab + b^2)$
$a^3 – b^3 = (a – b)(a^2 + ab + b^2)$

Formulas for the fourth degree

$(a + b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$
$(a – b)^4 = a^4 – 4a^3b + 6a^2b^2 – 4ab^3 + b^4$
$a^4 – b^4 = (a – b)(a + b)(a^2 + b^2)$

Practical work. Trigonometric formulas. Module Math.

Write a program that tests the formulas of abridged multiplication.

For all formulas, it is necessary to program the calculation of the right and left parts of the formula (if the part is not equal to a constant). Add the code to the next HTML page.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title> John Smith</title>
<script>
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
```

```
</script>
</head>
<body>
<div id="my">conteiner</div>
<script>
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
cnt.out("<font color='#0000FF' size='+3'> Trigonometric formulas.
Module Math.</font><br>");
gr_alfa = 67;
gr_beta = 11;
alfa = Math.PI/180*gr_alfa;
beta = Math.PI/180*gr_beta;
cnt.add("<b>Исходные данные</b><br>");
cnt.add("gr_alfa = "+gr_alfa+"<br>");
cnt.add("gr_beta = "+gr_beta+"<br>");
cnt.add("rd_alfa = "+alfa+"<br>");
cnt.add("rd_beta = "+beta+"<br>");
cnt.add("<br><b> Basic Trigonometric Formulas </b><br>");
tmp = Math.sin(alfa)*Math.sin(alfa)+Math.cos(alfa)*Math.cos(alfa);
cnt.add("sin^2(α)+cos^2(α) =
Math.sin(alfa)*Math.sin(alfa)+Math.cos(alfa)*Math.cos(alfa) =
"+tmp+"<br>");
tmp = 1+Math.tan(alfa)*Math.tan(alfa);
cnt.add("1+tg^2(α) = 1+Math.tan(alfa)*Math.tan(alfa) =
"+tmp+"<br>");
tmp = 1/Math.cos(alfa)/Math.cos(alfa);
cnt.add("1/cos^2(α) = 1/Math.cos(alfa)/Math.cos(alfa) =
"+tmp+"<br>");
cnt.add("<i> Basic Trigonometric Formulas </i><br>");
cnt.add("<br><h1><i> Add code for other sections </i></h1><br>");
</script>
```

```
</body>
</html>
```

Practical work. Logical Operations

Write a program that creates the following objects:
HTML_Container - output information to the container.
FORM_data - reading information from the form.
Triangle - a triangle, in this object should be the following elements:
- sides a, b, c;
- property tp - possible values: rectangular, obtuse, acute-angled, does not exist.
- methods init_tri, def_tp, toString (see the program below).

When you click the "compute" button, an instance of the "triangle" object is created and information about it is displayed (see the program below).

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title> John Smith </title>
<script>
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
```

```
}
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
    return this.frm.elements[nm].value;
  }
}
function Triangle(a, b, c) {
  this.a = Number(a);
  this.b = Number(b);
  this.c = Number(c);
  this.init_tri = function(){
    alert("sort the sides of the triangle, with should be the
largest ");
  }
  this.def_tp = function () {
    return " Return value rectangular, obtuse, acute, does not
exist ";
  }
  this.init_tri();
  this.tp = this.def_tp();
  this.toString = function () {
    return "Triangle : " + this.a + " " + this.b + " " + this.c + "
(" + this.tp + ")";
  }
}
</script>
</head>
<body>
<script>
function myCompute(){
    var elem = document.getElementById('my');
```

```
    var cnt = new HTML_Container(elem);
    var myf = new FORM_data("myForm");
    var a = myf.get("a");
    var b = myf.get("b");
    var c = myf.get("c");
    var tri = new Triangle(a, b, c);
    cnt.out("<font color='#0000FF' size='+3'>triangle</font><br>");
  cnt.add(tri+"<br>");
}
</script>
<form name="myForm">
a: <input name="a" type="text" value="5"><br>
b: <input name="b" type="text" value="4"><br>
c: <input name="c" type="text" value="3"><br>
<input name="compute" type="button" onClick="myCompute()"
value="compute">
</form>
<div id="my">conteiner</div>
</body>
</html>
```

Practical Work. Switch Statement.

Write a program that translates a number from one number system to another.

The program creates the following objects:

HTML_Container - output information to the container.

FORM_data - reading information from the form.

In the "Source data" field the initial number is written, the result is output to the container DIV.

The choice of the initial and final number system is carried out using the drop-down menu.

It is necessary to provide the following translation of numbers:

- "2->8"
- "2->10"
- "2->16"
- "8->2"
- "8->10"
- "8->16"
- "10->2"
- "10->8"
- "10->16"
- "16->2"
- "16->8"
- "16->10"

Pressing the "compute" button translates the number from one system to another and displays the result (see the program below).

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<style>
  .myclass {
    background-color: #CCFFFF;
    color: #CC0000;
    border: solid;
    border-color: #0000FF;
    width: 300px;
    height: auto;
}
</style>
<title> John Smith </title>
<script>
function HTML_Container(elem){
```

```
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
    return this.frm.elements[nm].value;
  }
}
</script>
</head>
<body>
<script>
function myCompute(){
  var elem = document.getElementById('my');
  var cnt = new HTML_Container(elem);
  var myf = new FORM_data("myForm");
  var act = myf.get("from_to");
  var src = myf.get("src");
  var res = 999999;
  var rout = "Error";
  switch (act) {
  case "2-8":
    res = parseInt(src,2)
```

```
    rout = res.toString(8);
    break;
  case "16-10":
    res = parseInt(src,16)
    rout = res.toString(10);
    break;
  default:
    alert( 'Error' );
}
  cnt.out(rout);
}
</script>
<form name="myForm">
Initial data:<input name="src" type="text" value="111001101">
// Add more translation options.
<select name="from_to">
  <option value="2-8">2-&gt;8</option>
  <option value="16-10">16-&gt;10</option>
</select>
<input name="compute" type="button" onClick="myCompute()"
value="compute">
</form>
Результат: <div class="myclass" id="my">conteiner</div>
</body>
</html>
```

Practical Work. Strings. Keyboard Layout.

Write a program that translates a string from one keyboard layout to another.

The program creates the following objects:

- FORM_data - reading information from form fields and writing information to form fields.
- SuperStr - methods of this object allow you to translate from one keyboard layout to another.

1. Create a page index.html.

- create a text field "Russian",
- create a text field "English",
- create a button "Rus-> Lat",
- create a button "Lat-> Rus",
- create a button "Clear".

2. The algorithm of operation.

When you press the "Rus-> Lat" button, the line from the "Russian" field is translated into the Latin layout (for example: "йцукен123" -> "qwerty123") and is written in the "English" field.

When you click the "Lat-> Rus" button, the line from the "English" field is translated into the Russian layout (for example: "qwerty123" -> "йцукен123") and is written in the field "Russian".

When the "Clear" button is pressed, the "English" and "Russian" fields are cleared (they contain blank lines).

It is necessary to supplement the code in the next program.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title>John Smith</title>
<script>
// read and write data form
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
```

```
    return this.frm.elements[nm].value;
  }
  this.set = function(nm,vl){
    this.frm.elements[nm].value = vl;
  }
}
// SuperStr constructor
function SuperStr(){
  var symRUS =
"йцукенгшщзхъфывапролджэячсмитьбюЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮ";
  var symLAT =
"qwertyuiop[]asdfghjkl;'zxcvbnm,.QWERTYUIOP{}ASDFGHJKL:\"ZXCVBNM<>"
;
  this.RusLat = function(strIn){
    var s = "", i, ch, n;
    for(i=0; i < strIn.length; i++){
    ch = strIn.charAt(i);
    n = symRUS.lastIndexOf(ch);
    if(n != -1) ch = symLAT.charAt(n);
    s += ch;
    }
    return s;
  }
  this.LatRus = function(strIn){
    alert("Create a translation method from the Latin keyboard
layout into Russian");
  }
}
</script>
</head>
<body>
<script>
```

```
// functions
function rus_lat(){
  var s0 = myf.get("inpRU");
  var s1 = sups.RusLat(s0);
  myf.set("inpEN",s1);
}
function lat_rus(){
  alert("Create a procedure for pressing the button Lat -> Rus ");
}
function clr_fld(){
  alert("Create a procedure for clearing input fields ");
}
</script>
<form name="myForm">
English: <input name="inpEN" type="text" value=""><br />
Russian: <input name="inpRU" type="text" value=""><br />
<input name="LatRus" type="button" onClick="lat_rus()" value="Lat -
> Rus">
<input name="Clr" type="button" onClick="clr_fld()" value="Clear">
<input name="RusLat" type="button" onClick="rus_lat()" value="Rus -
> Lat">
</form>
<script>
// Globals
var myf = new FORM_data("myForm");
var sups = new SuperStr();
</script>
</body>
</html>
```

Practical Work. Strings and Arrays. Transliteration

In the SuperStr object (See "Strings. Keyboard Layout"), add a method that transliterates the Russian character string.

It is necessary to supplement the code in the next program. All the necessary actions are described using the alert function.

Compare this program with "Strings. Keyboard Layout" program and find significant differences.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<style>
.myclass {
  background-color: #CCFFFF;
  color: #CC0000;
  border: solid;
  border-color: #0000FF;
  width: 300px;
  height: auto;
}
</style>
<title> John Smith</title>
<script>
// Global vars for SuperStr
var symRUS =
"йцукенгшщзхъфывапролджэячсмитьбюёЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮЁ
";
var symLAT =
"qwertyuiop[]asdfghjkl;'zxcvbnm,.`QWERTYUIOP{}ASDFGHJKL:\"ZXCVBNM<>
~";
arrTRL = [];
 arrTRL[0] = "y";   arrTRL[33] = "Y";   //й
```

```
 arrTRL[1] = "ts";  arrTRL[34] = "TS";  //ц
                                        //у
                                        //к
                                        //е
                                        //н
                                        //г
                                        //ш
                                        //щ
                                        //з
                                        //х
                                        //ъ
                                        //ф
                                        //ы
                                        //в
                                        //а
                                        //п
                                        //р
                                        //о
                                        //л
                                        //д
                                        //ж
                                        //э
                                        //я
                                        //ч
                                        //с
                                        //м
                                        //и
                                        //т
                                        //ь
                                        //б
arrTRL[31] = "yu";  arrTRL[64] = "YU";  //ю
arrTRL[32] = "yo";  arrTRL[65] = "YO";  //ё
```

```
alert("Fill all array arrTRL");

// End of global vars for SuperStr

// Write into container
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
// read and write data form
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
    return this.frm.elements[nm].value;
  }
  this.set = function(nm,vl){
    this.frm.elements[nm].value = vl;
  }
}
// SuperStr constructor
function SuperStr(){
  this.RusLat = function(strIn){
    var s = "", i, ch, n;
    for(i=0; i < strIn.length; i++){
```

```
      ch = strIn.charAt(i);
      n = symRUS.lastIndexOf(ch);
      if(n != -1) ch = symLAT.charAt(n);
      s += ch;
    }
    return s;
  }
  this.LatRus = function(strIn){
    var s = "", i, ch, n;
    for(i=0; i < strIn.length; i++){
      ch = strIn.charAt(i);
      n = symLAT.lastIndexOf(ch);
      if(n != -1) ch = symRUS.charAt(n);
      s += ch;
    }
    return s;
  }
  this.Translit = function(strIn){
    alert("Add code of translateration function ");
  }
}
</script>
</head>
<body>
<script>
// functions
function rus_lat(){
  var s0 = myf.get("inpRU");
  var s1 = sups.RusLat(s0);
  myf.set("inpEN",s1);
}
function lat_rus(){
```

```
  var s0 = myf.get("inpEN");
  var s1 = sups.LatRus(s0);
  myf.set("inpRU",s1);
}
function translit(){
  alert("Add code of event handler ");
}
function clr_fld(){
  myf.set("inpRU","");
  myf.set("inpEN","");
  alert("Add cleanup code for source code and transliteration
result ");
}
</script>
<form name="myForm">
English: <input name="inpEN" type="text" value=""><br />
Russian: <input name="inpRU" type="text" value=""><br />
<input name="LatRus" type="button" onClick="lat_rus()" value="Lat -
> Rus">
<input name="Clr" type="button" onClick="clr_fld()" value="Clear">
<input name="RusLat" type="button" onClick="rus_lat()" value="Rus -
> Lat"><br />
<br />Russian text for translit:<br />
<textarea name="Trans" cols="20" rows="5"></textarea><br />
<input name="Translit" type="button" onClick="translit()"
value="Translit">
</form>
<hr />
Translit result: <div class="myclass" id="my">conteiner</div>
<script>
// Globals
var elem = document.getElementById('my');
```

```
var cnt = new HTML_Container(elem);
var myf = new FORM_data("myForm");
var sups = new SuperStr();
</script>
</body>
</html>
```

Practical Work. Arrays. Simple Operations

The purpose of the work is to study and apply the methods of the Array object.

It is necessary to supplement the code in the next program. All the necessary actions are described using the alert function. As much as possible, use the methods of the Array object (rather than your own code).

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<style>
  .myclass {
    background-color: #CCFFFF;
    color: #CC0000;
    border: solid;
    border-color: #0000FF;
    width: 300px;
    height: auto;
}
</style>
<title> John Smith </title>
<script>
```

```
// Constructor of output container object
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}

// Constructor of read and write data form object
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
    return this.frm.elements[nm].value;
  }
  this.set = function(nm,vl){
    this.frm.elements[nm].value = vl;
  }
}
</script>
</head>
<body>
<script>
// functions
function process(){
  var s0 = myf.get("src_data");
  var arr = s0.split(" ");
```

```
  cnt.out("<b> Initial data</b><br>");
  cnt.add(arr+"<br>");

  cnt.add("<b> Number of entered numbers </b><br>");
  alert("Output to the container the number of entered numbers ");

  cnt.add("Sort Ascending </b><br>");
  var tmp = [];
  alert("Display the result of sorting in ascending order ");

  cnt.add("<b> Sorting in decreasing order </b><br>");
  alert("Display the result of sorting in descending order ");

  cnt.add("<b>Minimum </b><br>");
  alert("Output to the container the minimum element ");

  cnt.add("<b> Maximum </b><br>");
  alert("Display the maximum element in the container ");

  cnt.add("<b> Sum of elements </b><br>");
  alert("Display the sum of the elements in the container ");

  cnt.add("<b> Average value </b><br>");
  alert("Output to the container the average value of the elements
");

}
function clr_fld(){
  alert("Clear input and output fields ");
}
</script>
```

```
<form name="myForm">
Array elements, separated by spaces:<br />
<textarea name="src_data" cols="40" rows="5">10 9 7 8 5 6 1 2 3
4</textarea><br />
<input name="Process" type="button" onClick="process()" value="Data
processing">
<input name="Clr" type="button" onClick="clr_fld()" value="Clear">
</form>
<hr />
Result: <div class="myclass" id="my">conteiner</div>
<script>
// Globals
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
var myf = new FORM_data("myForm");
</script>
</body>
</html>
```

Practical Work. Arrays and Objects

The purpose of the work is to study and apply in practice the compilation and use of complex hierarchical structures. Structures are compiled using a combination of objects and arrays.

Write a program for calculating and outputting your estimates for the course "Scripting programming languages", for this you need to supplement the code in the following program.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title> John Smith</title>
```

```
<style>
td{
  border-collapse: collapse;
  border-color: #999999;
  border-style: solid;
  border-width: 1px;
}
table{
  border-collapse: collapse;
  border-color: #999999;
  border-style: solid;
  border-width: 1px;
}
.css_total {
  background-color: #CCFFFF;
  color: #CC0000;
  font-weight: bold;
}
.css_modul{
  background-color: #DBDBDB;
}
.css_task{
  background-color: #F9FFF9;
}
</style>
<script>
//Global array
var MyTasks =
  {nam:" Final Grade", cls:"css_total", mgr:100, rgr:0,
    mdl:[
          {nam:"Mod №1", cls:"css_modul", mgr:35, rgr:0,
            tsk:[
```

```
                {nam:"LW №1.1", cls:"css_task", mgr:5, rgr:0},
                {nam:"LW №1.2", cls:"css_task", mgr:5, rgr:0}
              ]
        },
        {nam:"Mod №2",  cls:"css_modul", mgr:35, rgr:0,
          tsk:[
                {nam:"LW №2.1", cls:"css_task", mgr:5, rgr:0},
                {nam:"LW №2.2", cls:"css_task", mgr:5, rgr:0}
              ]
        },
        {nam:" Exam ", cls:"css_modul", mgr:30, rgr:0,
          tsk:[
                {nam:"Ind №1", cls:"css_task", mgr:5, rgr:0},
                {nam:"Ind №2", cls:"css_task", mgr:5, rgr:0}
              ]
        }
      ]
  }
// Constructor of output container object
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
```

```
// Constructor of read and write data form object
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
    return this.frm.elements[nm].value;
  }
  this.set = function(nm,vl){
    this.frm.elements[nm].value = vl;
  }
}
</script>
</head>
<body>
<script>
// functions
function compute_marks(){
  var i,j,n,k,rgr,mgr;
  var s0 = myf.get("src_data");
  var arr = s0.split(" ");
  n=0;
  for(i=0; i<MyTasks.mdl.length; i++){
    rgr = 0;
    mgr = 0;
    for(j=0; j< MyTasks.mdl[i].tsk.length; j++){
      k=0;
      if(n<arr.length) k=Number(arr[n++]);
      MyTasks.mdl[i].tsk[j].rgr = k;
      rgr += k;
      mgr += MyTasks.mdl[i].tsk[j].mgr;
    }
    MyTasks.mdl[i].rgr=Math.round(rgr*MyTasks.mdl[i].mgr/mgr);
    // MyTasks.mdl[i].rgr=rgr*MyTasks.mdl[i].mgr/mgr;
```

```
    MyTasks.rgr += MyTasks.mdl[i].rgr;
  }
}

function out_marks(){
  var i,j,s;
  cnt.clr();
  s = "<table border=1>";
  s += "<tr class='"+MyTasks.cls+"'>";
  s += "<td>"+MyTasks.nam+"</td>";
  s += "<td>"+MyTasks.mgr+"</td>";
  s += "<td>"+MyTasks.rgr+"</td>";
  s += "</tr>";
  for(i=0; i<MyTasks.mdl.length; i++){
    s += "<tr class='"+MyTasks.mdl[i].cls+"'>";
    s += "<td>"+MyTasks.mdl[i].nam+"</td>";
    s += "<td>"+MyTasks.mdl[i].mgr+"</td>";
    s += "<td>"+MyTasks.mdl[i].rgr+"</td>";
    s += "</tr>";
    for(j=0; j< MyTasks.mdl[i].tsk.length; j++){
      s += "<tr class='"+MyTasks.mdl[i].tsk[j].cls+"'>";
      s += "<td>"+MyTasks.mdl[i].tsk[j].nam+"</td>";
      s += "<td>"+MyTasks.mdl[i].tsk[j].mgr+"</td>";
      s += "<td>"+MyTasks.mdl[i].tsk[j].rgr+"</td>";
      s += "</tr>";
    }
  }
  s += "</table>";
  cnt.add(s);
}

function process(){
```

```
  compute_marks();
  out_marks();
}

function clr_marks(){
  var i,j;
  for(i=0; i<MyTasks.mdl.length; i++){
    for(j=0; j< MyTasks.mdl[i].tsk.length; j++){
      MyTasks.mdl[i].tsk[j].rgr = 0;
    }
    MyTasks.mdl[i].rgr= 0;
  }
  MyTasks.rgr = 0;
  out_marks();
}
</script>
<form name="myForm">
All my marks, separated by spaces:<br />
<textarea name="src_data" cols="40" rows="5">0 1 2 3 4 5 0 1 2 3 4
5</textarea><br />
<input name="Process" type="button" onClick="process()" value="Data
processing">
<input name="Process" type="button" onClick="clr_marks()"
value="Clear marks">
</form>
<hr />
Result: <div class="myclass" id="my">conteiner</div>
<script>
// Globals
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
var myf = new FORM_data("myForm");
```

```
</script>
</body>
</html>
```

References:

1. Бочаров Б.П. Інформаційні технології в освіті : монографія / Б.П. Бочаров, М.Ю. Воєводіна; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків: ХНУМГ ім. О. М. Бекетова, 2015. – 197 с.

2. Bocharov Boris. AUTOMATIZED WEB PAGES PARSING AND CREATION / Boris Bocharov, Maria Voevodina // Information technologies in education: electronic supplement to the journal "Educational Institutions Libraries". – 2017. – N5, p. 1-5.